Test-Driven Development (TDD) for JavaScript & Python

Test-Driven Development (TDD) is a software development approach where tests are written before the actual code. The development process follows a cycle: write a test, run the test (it should fail initially), write the code to pass the test, and then refactor if needed. TDD ensures that your code meets the requirements and remains maintainable.

To apply TDD in our **E-Commerce System** project, we use **Jest** as the testing framework for frontend and **Pytest** for backend.

Setting Up the Environment:

Frontend - Jest Setup:

1. Install Jest in the React project:

   npm install --save-dev jest @testing-library/react @testing-library/jest-dom

2. Update package.json with the following script:

```
{
    "scripts": {
      "test": "jest"
    }
  }
```

3. Create a Jest configuration file, e.g., jest.config.js:

```
module.exports = {
    moduleNameMapper: {
        testEnvironment: 'jsdom',
        setupFilesAfterEnv: ['./jest.setup.js'],
    },
  };
```

Backend - Pytest Setup:

1. Install pytest for your Django project:

   pip install pytest

2. Create a **tests** directory in your Django app.

3. Write a simple **pytest.ini** configuration:

```
[pytest]
DJANGO_SETTINGS_MODULE = your_project.settings
```

4. Create a **conftest.py** file in the **tests** directory for fixtures and configuration.

TDD Workflow in Django-React Project:

Frontend TDD with Jest:

1. Write a failing test(SumComponent.test.js) for a React component. For example:

```
import React from 'react';
import { render } from '@testing-library/react';
import SumComponent, { sum } from './SumComponent';

test('adds 1 + 2 to equal 3', () => {
  expect(sum(1, 2)).toBe(3);
});

test('renders sum component correctly', () => {
  const { getByText } = render(<SumComponent a={1} b={2} />);
  const resultElement = getByText('The sum of 1 and 2 is: 3');
  expect(resultElement).toBeInTheDocument();
});
```

2. Run the test using npm test. It should fail initially.
3. Write the minimal code (SumComponent.js) to make the test pass:

```jsx
import React from 'react';

export function sum(a, b) {
  return a + b;
}

function SumComponent(props) {
  const { a, b } = props;
  const result = sum(a, b);

  return <div>The sum of {a} and {b} is: {result}</div>;
}

export default SumComponent;
```

4. Run the test again to ensure it passes.

Backend TDD with Pytest:

1. Write a failing test for a Django view or API. For example:

```python
# test_views.py

def test_sum_view(client):
    response = client.get('/sum/?a=1&b=2')
    assert response.status_code == 200
    assert response.json()['result'] == 3
```

2. Run the test using pytest. It should fail initially.
3. Write the minimal code to make the test pass:

```python
# views.py

from django.http import JsonResponse

def sum_view(request):
    a = int(request.GET.get('a', 0))
    b = int(request.GET.get('b', 0))
    result = a + b
    return JsonResponse({'result': result})
```

4. Run the test again to ensure it passes

Implementing Test-Driven Development in your Django-React project using Jest for frontend and Pytest for backend helps ensure the reliability and correctness of our codebase.